HEWLETT-PACKARD COMPANY
Legal Department, 20BN
P.O. Box 10301
Palo Alto, California 94303-0890

PATENT APPLICATION

ATTORNEY DOCKET NO. 10981786-1

IN THE U.S. PATENT AND TRADEMARK OFFICE
Patent Application Transmittal Letter

ASSISTANT COMMISSIONER FOR PATENTS
Washington, D.C. 20231

Sir:

Transmitted herewith for filing under 37 CFR 1.53(b) is a(n): (X) Utility ( ) Design

(X) original patent application,

( ) continuation-in-part application

INVENTOR(S): **Ding-Kai Chen, Dz-Ching Ju**

TITLE: **SYSTEM AND METHOD FOR EFFICIENTLY PASSING INFORMATION BETWEEN COMPILER AND POST-COMPILE-TIME SOFTWARE**

Enclosed are:

(X) The Declaration and Power of Attorney. (X) signed ( ) unsigned or partially signed

(X) 9 sheets of drawings (one set) ( ) Associate Power of Attorney

( ) Form PTO-1449 ( ) Information Disclosure Statement and Form PTO-1449

( ) Priority document(s) ( )(Other) _____ (fee $_____)

CLAIMS AS FILED BY OTHER THAN A SMALL ENTITY				
(1) FOR	(2) NUMBER FILED	(3) NUMBER EXTRA	(4) RATE	(5) TOTALS
TOTAL CLAIMS	16 — 20	0	X $18	$ 0
INDEPENDENT CLAIMS	3 — 3	0	X $78	$ 0
ANY MULTIPLE DEPENDENT CLAIMS	0		$260	$ 0
BASIC FEE: Design $310.00); Utility $760.00)				$ 760
TOTAL FILING FEE				$ 760
OTHER FEES				$
TOTAL CHARGES TO DEPOSIT ACCOUNT				$ 760

Charge $ 760 to Deposit Account 08-2025. At any time during the pendency of this application, please charge any fees required or credit any over payment to Deposit Account 08-2025 pursuant to 37 CFR 1.25. Additionally please charge any fees to Deposit Account 08-2025 under 37 CFR 1.16, 1.17,1.19, 1.20 and 1.21. A duplicate copy of this sheet is enclosed.

"Express Mail" label no. EL246450721US

Date of Deposit 10-21-99

I hereby certify that this is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and is addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231.

By Evelyn Sanders

Typed Name: **Evelyn Sanders**

Respectfully submitted,

Ding-Kai Chen, Dz-Ching Ju

By _____

Robert E. Stachler, II

Attorney/Agent for Applicant(s)

Reg. No. 36,934

Date:

Telephone No.: (770) 933-9500

SYSTEM AND METHOD FOR EFFICIENTLY PASSING INFORMATION BETWEEN COMPILER AND POST-COMPILE-TIME SOFTWARE

BACKGROUND OF THE INVENTION

FIELD OF THE INVENTION

The present invention is generally related to program optimization, and more particularly related to an apparatus and method for efficiently passing compiler information to post-compile-time software.

DESCRIPTION OF RELATED ART

As is known in the computer and software arts, when a software program is developed it will be optimized to run on a particular computer architecture. While it is possible that the software program developed for an original computer architecture will run on a computer system with a new or different architecture, the execution of the software program optimized for an old computer architecture will not generally run as quickly on a computer system with a new architecture, if at all. Therefore, devising a way to run an existing (i.e. old) architecture binary version of a computer program on a new architecture or improve the performance of the computer program on the existing architecture, is an important procedure. One such way to

improve the performance of a computer program is to utilize a post-compile-time dynamic optimizer.

When software tools such as dynamic optimizer or profiling tools work on the binaries produced by the compiler, they face the challenge of analyzing low-level programs. It is desirable that some information can be passed from the compiler to the dynamic optimizer to make the analysis easier and more efficient.

A good example is that when a dynamic optimizer generates code at run-time, it often needs to perform register liveness analysis of the binary code in order to find unused registers that can be used without altering the program behavior. Liveness of a register occurs when the register contains data that is to be utilized in subsequent processing. A register can switch between active usage of storing a value for later consumption (live), and an inactive state (dead). Since the compiler has already performed a liveness analysis, reusing this liveness information is just a matter of how to pass the information to the dynamic optimizer efficiently.

Heretofore, software developers have lacked an apparatus and method for passing compile time information at run time to post-compile-time software in an efficient way.

SUMMARY OF THE INVENTION

To achieve the advantages and novel features, the present invention is generally directed to a system and method for efficiently passing compiler information at run time to hardware or software in an efficient way. The present invention is particularly useful for efficiently passing compiler information during code optimization or translation utilizing free or unused operand fields of instructions such as NOP, and encoding the compile time information in the unused operand field. This technique removes the time overhead for analyzing binaries or low-level programs and does not increase program code size.

The present invention provides a system and method for passing compile time information between a compiler and real-time operation of post-compile-time software. Briefly described, in architecture, the system can be implemented as follows. The preferred system of the present invention utilizes an unused NOP operand (a register usage bit vector) that is a vehicle (or communication channel) between a static compiler and a dynamic optimizer. Each bit in the vector represents a particular register and is used to indicate if the register may be live. The register usage bit vector in the unused NOP operand is used to make finding free registers easier during optimization.

The present invention can also be viewed as providing a method for passing compile time information between a compiler and real-time operation of post-compile-time software. In this regard, the method can be broadly summarized by the following steps: the compiler produces bit vectors for each basic block, (*i.e.*, subroutine, function, and/or procedure) and places the bit vector in the unused portion of the NOP instruction encoding. A bit in the vector represents a particular register. A bit is set if the register may be live at the location of the NOP instruction and allows the dynamic optimizer to determine if further analysis of the low-level code to determine whether the register is truly live is required. On the other hand, a zero (*i.e.*, unset) bit in the bit vector signals that the compiler does not use the corresponding register (*i.e.*, is a dead register) at the location of the NOP instruction, and therefore the register can be used by the dynamic optimizer.

NOP instructions perform no operation and are generally used as filler or instruction place-holders. For example, NOP operations have an immediate operand that is not used. These operand fields provide an efficient one-way communication channel between the compiler and hardware or software.

Because the compiler stores the dead register information in the unused operand area, the analysis information can be accessed without making the low-level code larger. Should the low-level code

not have a NOP instruction to encode register utilization information, the dynamic optimizer or other software can examine the operands of NOPs instructions in the surrounding basic blocks to deduce the missing information. A basic block is a collection of a sequence of instructions that are entered at the top of the sequence and exited at the bottom of the sequence.

An advantage of deducing this missing information is that the information is particularly useful in improving performance of dynamic optimizations performed at runtime. This is because the analysis overhead directly reduces performance when performed. In the preferred method of the present invention, because the dynamic optimizations may inspect the unused NOP operands very quickly, the overhead is dramatically reduced to improve runtime performance.

In another embodiment, the compiler may pass hints to profiler software of what kind of feedback information is desired through the use of the unused NOP operands. Because of many other possible uses of this communication channel, the compiler has to annotate the low-level code binaries it produces to indicate what information is contained in the unused NOP operands.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings incorporated in and forming a part of the specification illustrate several aspects of the present invention,

and together with the description, serve to explain the principles of the invention. In the drawings:

FIG. 1 is a block diagram of a user system showing the compiler and dynamic optimizer of the present invention within the memory area.

FIG. 2A is a block diagram of a possible example of a bundled instruction, having three instructions grouped together in an X bit sized and aligned container.

FIG. 2B is a block diagram of a possible example of a NOP instruction.

FIGs. 3A is a block diagram of the system of the present invention showing the interaction between the code register usage annotator and the system compiler to create a binary program code from the original source code.

FIG. 3B is a block diagram of the system of the present invention showing the interaction between the binary program code, the dynamic optimizer, and the instruction vector comparator to create new optimized binary program code.

FIG. 4 is a flow chart of the preferred method to perform the compilation process, as shown in FIG. 1 and utilizing a possible example of NOP instructions.

FIG. 5 is a flow chart of an example of the preferred method to perform the code register usage annotation process, as shown in FIG 3A.

FIG. 6 is a block diagram showing the bit vector generated for each basic block of the present invention utilizing a possible example of NOP instructions as shown in FIG. 2B.

FIG. 7 is a flow chart of the preferred method to perform the dynamic optimization process of the present invention that utilizes the bit vectors to indicate register usage.

FIG. 8 is a flow chart of the preferred method to perform the free register information process of the present invention that utilizes the bit vector to indicate register usage.

FIG. 9 is a block diagram of a possible example 130 of utilization of the present invention with regard to multiple basic blocks and multiple registers.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Reference will now be made in detail to the description of the invention as illustrated in the drawings. Although the invention will be described in connection with these drawings, there is no intent to limit it to the embodiment or embodiments disclosed therein. On the contrary, the intent is to include all alternatives, modifications, and

equivalents included within the scope of the invention as defined by the appended claims.

As illustrated in FIG. 1A, computer system 12 today generally comprises a processor 21 and memory 31 (*e.g.*, RAM, ROM, hard disk, CD-ROM, etc.) including an operating system 32. The processor 21 accepts binary program code 62 and data from the memory 31 over the local interface 23, for example, a bus(es). Direction from the user can be signaled by using input devices, for example but not limited to, a mouse 24 and a keyboard 25. The actions input and result output are displayed on the display terminal 26.

Also shown is the compiler 60, binary program code 62, translated binary program code 130 and dynamic optimizer 100 in memory area 31. These components and their operation are herein described in further detail with regard to FIGS. 4-9.

Illustrated in FIG. 2A is a block diagram of an example of three instructions that are grouped together in an X-bit sized and aligned container called a bundle 41. Each bundle 41 contains three Y-bit instruction slots 42-44, a Z-bit template 45 and a stop bit fields. The stop bit field 46 specifies whether the instruction group boundary occurs after the last instruction of the current bundle 41. Each of the instruction slots 42-44 includes an op code and necessary operands to execute one instruction. The template 45 specifies two properties: the instruction group boundaries within the current bundle 41, and

the mapping of the instruction slots to execution unit types. Within a bundle 41, execution order proceeds from slot 0 to slot 2.

The inventors have described an example of a specific instruction bundle format. However, the inventors contemplate that the present invention can be applied to system architectures that do not bundle instructions or bundle instructions in a different way from that disclosed above.

Illustrated in FIG. 2B is an example of a possible instruction format for NOP instruction 50. As illustrated, the opcode bits 51 occupy bits 40 through 37 of the possible NOP instruction. A reserved bit 52 occupies bit 36 of the possible instruction format for NOP instruction 50. Opcode extensions 53 occupy bits 35 through 27 of a possible instruction format for a NOP instruction 50. Unused bits 56 occupy bits 26 through 6 of a possible instruction format for NOP instruction 50. This is where the register usage bit vector of the present invention is stored. Bits 5 through 0 are reserved bits 57. The inventors further contemplate that the present invention can be utilized with any system architecture, as long as there are unused bits in a NOP instruction 50, regardless of the particular instruction format.

Illustrated in FIG. 3A is a block diagram showing the interaction of the code register usage annotator 70, for the system compiler 60, and the created binary object code 62 of the present invention. The

original source code 61 is input into the system compiler 60 of the present invention.

The system compiler 60 of the present invention includes the improvement that interacts with the code register usage annotator 70, for bit vector generation. This is done to assist the dynamic optimizer 100 in the creation of the new optimized binary object code 130. The system compilation process 60 is herein defined in further detail with regard to FIG. 4.

The code register usage annotator 70 generates bit vector annotation for each NOP instruction in the binary code based on register allocation results of the system compiler 60. The code register usage annotator 70 is herein defined in further detail with regard to FIG. 5.

Illustrated in FIG. 3B is a block diagram illustrating the dynamic optimizer process of the present invention. As illustrated in FIG. 3B, the binary object code 62 generated by the system compiler 60 of the present invention is input into the dynamic optimizer 100 of the present invention.

The dynamic optimizer 100 interacts with the instruction bit vector comparator 120 for each NOP instruction in each basic block to generate the newly optimized binary object code 130. The instruction bit vector comparator 120 utilizes the bit vectors created in NOP instructions in each basic block by the code register usage annotator

70 process. The dynamic optimizer process illustrated in FIG. 3B is herein defined in further detail with regard to FIGs. 7 and 8.

Illustrated in FIG. 4 is the compilation process 60 of the present invention. First, the compilation process is initialized at step 61. Next, the compilation process 60 performs a lexical analyzer at step 62. Then, a parser is executed at step 63. The parser is a process that processes the sequence of tokens and produces an intermediate level representation, such as a parse tree or sequential intermediate code and symbol table, that records the identifiers used in the program and/or attributes. The parser may produce error messages if the token strings contain syntax errors.

The semantic analyzer operation is performed at step 64. The semantic analyzer is for checking a program for validity. This process takes the input of the intermediate code generated in the parsing step 63 and a symbol table, and determines whether the program satisfies the schematic properties required by the source language, *i.e.*, where the identifiers are consistently declared and used. The semantic analyzer step 64 may produce an error message if the program is semantically inconsistent or fails in some other way to satisfy the requirements of the programming language definitions.

The register allocations are then performed at step 65. Then, the compilation process 60 performs the code generation process at step 66. Code generation utilizes the intermediate code generated in

the parser step 63 and semantic analyzer step 64 and transforms the code into equivalent machine code in a form of a relocatable object module or directly executable object code. Any detected errors may be warnings or definite errors and in the later case may terminate the compilation.

Then, the code register usage annotation process of the present invention is performed at step 67. The code register usage annotation process is herein defined in further detail with regard to FIG. 5.

Next, the compilation process performs the final assembly process at step 68. However, this step is optional since many compilers generate binary machine codes without requiring an assembly output. The compilation process 60 is then exited at step 69.

An alternative approach involves the code register usage annotation step 67. It is contemplated by the inventors that the compilation process 60 can inherit the register allocation information generated at step 65 and use this information since the compiler has determined what registers have been used and where the registers are used already. In this way, the compiler process 60 would not need to again scan the code to determine the register usage, as shown in FIG. 5.

Illustrated in FIG. 5 is a flow chart of the process for determining the code register usage annotation 70. The code register

usage annotation process 70 is first initialized at step 71. The code register usage annotation process 70 then gets the next basic block (*i.e.*, subroutine, function, or procedure) for analysis at step 72. The code register usage annotation process 70 determines the registers used in each basic block at step 73.

Next, the code register usage annotation process 70 locates a NOP instruction in the basic block at step 74. At step 75, the code register usage annotation process 70 creates a register usage bit vector 80 in the unused area of the NOP instruction located at step 74. The code register usage annotation process 70 sets all the bits in the NOP instruction register usage bit vector 80 corresponding to each caller-save register that is live at the location of the NOP instruction in the basic block at step 76. Liveness of a register occurs when the register contains data that is to be utilized in subsequent processing. A register can switch between active usage of storing a value (live) for later consumption, and an inactive state (dead).

At step 77, the code register usage annotation process 70 determines whether all NOP instructions in the basic block have been processed. If all NOP instructions in the basic block have not been processed at step 77, the code register usage annotation process 70 then returns to repeat step 74 through 77.

If the code register usage annotation process 70 has processed all NOP instructions in the basic block, the code register usage

annotation process then in step 78, determines whether all the basic blocks have been processed for NOP instructions. If the code register usage annotation process 70 determine that all the basic blocks have not been processed, the code register usage annotation process 70 returns to repeat step 72 through 78. If the code register usage annotation process 70 has processed all NOP instructions of all the basic blocks in the original source program 34 (FIG. 3A), the code register usage annotation process 70 exits at step 79.

Illustrated in FIG. 6 is a block diagram representing an example of the structure of the code register bit vector 80 contained within the example of a NOP instruction's unused bit 56 ranging from 26 to 6. A total of 21 register usage bits of the example NOP instruction are available for register usage indication. The register usage bit vector 80 comprises a plurality of register usage bits 81A through 81U. Each caller saved register has a corresponding usage bit within the register usage bit vector 80 contained within the NOP instruction's unused bits 56 ranging from 26 to 6.

Illustrated in FIG. 7 is a flow chart of the dynamic optimizer process 100 of the present invention. First, the dynamic optimizer process 100 is initialized at step 101. The initialization process 100 determines or finds all the source code entry points at step 102. The dynamic optimizer process 100 analyzes the binary object code 62 from each entry point as determined in step 102, to ascertain all the

instructions and storage areas, including registers and memory locations utilized by the binary object code 62.

At step 104, the dynamic optimizer process 100 performs the free register information process 110. The free register information process 110 is herein defined in further detail with regard to FIG. 8. After the free register information process 110 has been performed, the dynamic optimizer process 100 translates the created binary object code 62 into the new optimized binary object code 130 at step 105. The dynamic optimizer process then exits at step 109.

Illustrated in FIG. 8 is the flow chart for the free register information process 110. The free register information process 110 is first initialized at step 111. The free register information process 110 retrieves all register usage bit vectors 80 in the affected NOP instructions, from all basic blocks within the binary object code 62 at step 112. For each NOP instruction, the register usage bit vector 80 indicates the register usage at the location of the NOP instruction containing the register usage bit vector 80. The register usage bit vector 80 of the basic block containing the NOP instruction is deduced from the register usage bit vector 80, at step 113. A basic block is a collection of a sequence of instructions that are entered at the top of the sequence and exited at the bottom of the sequence.

At step 114, the free register information process 110 determines if optimization is to be performed across basic block

boundaries. If so, the free register information process 110 performs the bit-OR operation on the register usage bit vectors 80 of the affected basic blocks. This is done to determine which of the registers are not in use in any of the basic blocks. The free register information process 110 utilizes the register usage bit vectors 80 of the affected basic block instructions to optimize the binary object code 62 across basic block boundaries on-demand at step 115. Step 115 is not performed unless an optimization involves more than one basic block, and then only needs to logically "OR" the register usage bit vectors 80 of the basic blocks involved in one optimization.

If the free register information process 110 determines at step 114 that it is not configured to optimize across basic block boundaries, the free register information process 110 proceeds to step 116 to utilize the free registers available for optimization. After utilizing the free registers available for optimization, the free register information process 110 exits at step 119.

Illustrated in FIG. 9 is a block diagram of a possible example 130 of the utilization of the present invention with regard to 4 basic blocks (131-134) and the register usage of registers R2, R3 and R4. As shown in basic block 131, a NOP instruction is encountered and encoded prior to the loading of any registers. Therefore, the NOP instruction is encoded in the register usage bit vector 80 to reflect that all registers are dead. Basic block 131 contains code that loads data

into registers R2 and R3 and then jumps to basic block 132 or basic block 133 for further processing.

Basic block 132 contains code that utilizes register R2 and causes register R2 to change status from live to dead. The NOP instruction in basic block 132 is encoded to reflect that registers R2 and R4 are dead, and that register R3 is live. This indicates that the contents of register R2 were consumed.

As shown in basic block 133, a NOP instruction is encountered and encoded prior to the processing of any registers. The NOP instruction in basic block 133 is encoded to reflect that registers R2 and R3 are live and that register R4 is dead.

Basic block 134 contains code that loads registers R2 and R4 and causes a change of state for registers R2 and R4 from dead to live. Basic block 134 further contains code that utilizes data in register R3. The NOP instruction in basic block 134 is encoded to reflect that R2 and R4 are live, and that R3 is dead.

The optimization using unused operands in the NOP instruction system comprises an ordered listing of executable instructions for implementing logical functions, can be embodied in any computer-readable medium for use by or in connection with an instruction execution system, apparatus, or device, such as a computer-based system, processor-containing system, or other system that can fetch the instructions from the instruction execution system, apparatus, or

device and execute the instructions. In the context of this document, a "computer-readable medium" can be any means that can contain, store, communicate, propagate, or transport the program for use by or in connection with the instruction execution system, apparatus, or device.

The computer readable medium can be, for example but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, device, or propagation medium. More specific examples (a nonexhaustive list) of the computer-readable medium would include the following: an electrical connection (electronic) having one or more wires, a portable computer diskette (magnetic), a random access memory (RAM) (magnetic), a read-only memory (ROM) (magnetic), an erasable programmable read-only memory (EPROM or Flash memory) (magnetic), an optical fiber (optical), and a portable compact disc read-only memory (CDROM) (optical).

Note that the computer-readable medium could even be paper or another suitable medium upon which the program is printed, as the program can be electronically captured, via for instance, optical scanning of the paper or other medium, then compiled, interpreted or otherwise processed in a suitable manner if necessary, and then stored in a computer memory.

The foregoing description has been presented for purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obvious modifications or variations are possible in light of the above teachings. The flow charts of the present invention show the architecture, functionality, and operation of a possible implementation of the register usage optimization compilation and translation system. In this regard, each block represents a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that in some alternative implementations, the functions noted in the blocks may occur out of the order noted in the figures, or for example, may in fact be executed substantially concurrently or in the reverse order, depending upon the functionality involved.

The embodiment or embodiments discussed were chosen and described to provide the best illustration of the principles of the invention and its practical application to enable one of ordinary skill in the art to utilize the invention in various embodiments and with various modifications as are suited to the particular use contemplated. All such modifications and variations are within the scope of the invention as determined by the appended claims when interpreted in accordance with the breadth to which they are fairly and legally entitled.

CLAIMS

What is claimed is

1. A register usage indicator system for efficiently signaling register usage in a computer program comprising a plurality of blocks of code, said register usage indicator system comprising:

a code usage register contained within a NOP instruction in one of the plurality of blocks of code in the computer program, said code usage register comprising a plurality of storage bits; and

a code register usage annotator for determining if each one of the plurality of registers is live in one of the plurality of blocks of code containing said NOP instruction.

2. The system of claim 1, wherein said code register usage annotator sets one of said plurality of storage bits in said code usage register for each one of the plurality of registers that is live in one of the plurality of blocks of code containing said NOP instruction.

3. The system of claim 1, further comprising:

a register usage comparator for determining which of said registers are live in one of the plurality of blocks of code in the computer program by inspecting the bits set in said code usage register contained in said NOP instruction.

4. The system of claim 3, wherein said code register usage annotator determines whether or not each register is live in each one of the plurality of blocks of code containing said NOP instruction; and

wherein said code register usage annotator sets each one of the plurality of storage bits in one of a plurality of storage code usage registers for each register live in each one of the plurality of blocks of code containing said NOP instruction.

5. The system of claim 4, wherein said register usage comparator determines which of said registers are not live in one of said plurality of blocks of code, by performing a logical OR of all of said plurality of storage code usage registers.

6. A method to efficiently signal register usage in a computer program comprising a plurality of blocks of code, the method comprising the steps of:

determining which of a plurality of registers are live in one of the plurality of blocks of code in the computer program;

finding at least one NOP instruction in one of the plurality of blocks of code;

creating a code usage register having a plurality of storage bits in said at least one NOP instruction in one of the plurality of blocks of code; and

setting one of said plurality of storage bits for each one of the plurality of registers live in one of the plurality of blocks of code containing said NOP instruction.

7. The method of claim 6, wherein said determining step further comprises the step of:

determining which of said plurality of registers are live in one of the plurality of blocks of code by inspecting the bits set in said code usage register.

8. The method of claim 7, further comprising the step of:

determining which of the plurality of registers is live in each one of the plurality of blocks of code in the computer program.

9. The method of claim 8, further comprising the step of:

setting each one of said plurality of storage bits in one of a plurality of storage code usage registers for each register live in one of the plurality of blocks of code containing said NOP instruction.

10. The method of claim 9, further comprising the step of:

determining which of said registers are not live in all of the plurality of blocks of code, in the computer program, by performing a logical OR of all of said plurality of storage code usage.

11. A register usage indicator system for efficiently signaling register usage in a computer program comprising a plurality of blocks of code, said register usage indicator system comprising:

means for determining which of the plurality of registers are live in one of the plurality of blocks of code in the computer program;

means for finding at least one NOP instruction in said one of the plurality of blocks of code;

means for creating a code usage register in said at least one NOP instruction in said one of the plurality of blocks of code; and

means for setting one of a plurality of storage bits in said code usage register for each one of the plurality of registers live in said one of the plurality of blocks of code containing said NOP instruction.

12. The system of claim 11, further comprising:

means for determining which of the plurality of registers are live in each one of the plurality of blocks of code in the computer program.

13. The system of claim 12, wherein said determining means further comprises:

means for inspecting the bits set in said code usage register to determine which of said registers are live in one of the plurality of blocks of code containing said NOP instruction.

14. The system of claim 13, further comprising:

means for determining which of the plurality of registers are live in each one of the plurality of blocks of code in the computer program.

15. The system of claim 14, further comprising:

means for setting each one of said plurality of storage bits in one of a plurality of storage code usage registers for each register live in each one of the plurality of blocks of code in the computer program.

16. The system of claim 15, further comprising:

means for determining which of said registers are not live in any of the plurality of blocks of code in the computer program, by performing a logical OR of all of said plurality of storage code usage registers.

ABSTRACT OF THE DISCLOSURE

System and method are described for register optimization during code translation utilizes a technique that removes the time overhead for analyzing register usage and eliminates fixed restraints on the compiler register usage. The present invention for register optimization utilizes a compiler to produce a register usage bit vector in a NOP instruction within each basic block (*i.e.*, subroutine, function, and/or procedure). Each bit in the bit vector represents a particular caller-saved register. A bit is set if, at the location of NOP instruction, the compiler uses the corresponding register within that basic block containing the NOP instruction to hold information to be used at a later time. During the translation, the translator examines the register usage bit vector to very quickly determine which registers are free and therefore can be used during the register optimization without the need to save and restore the register values.

FIG. 1

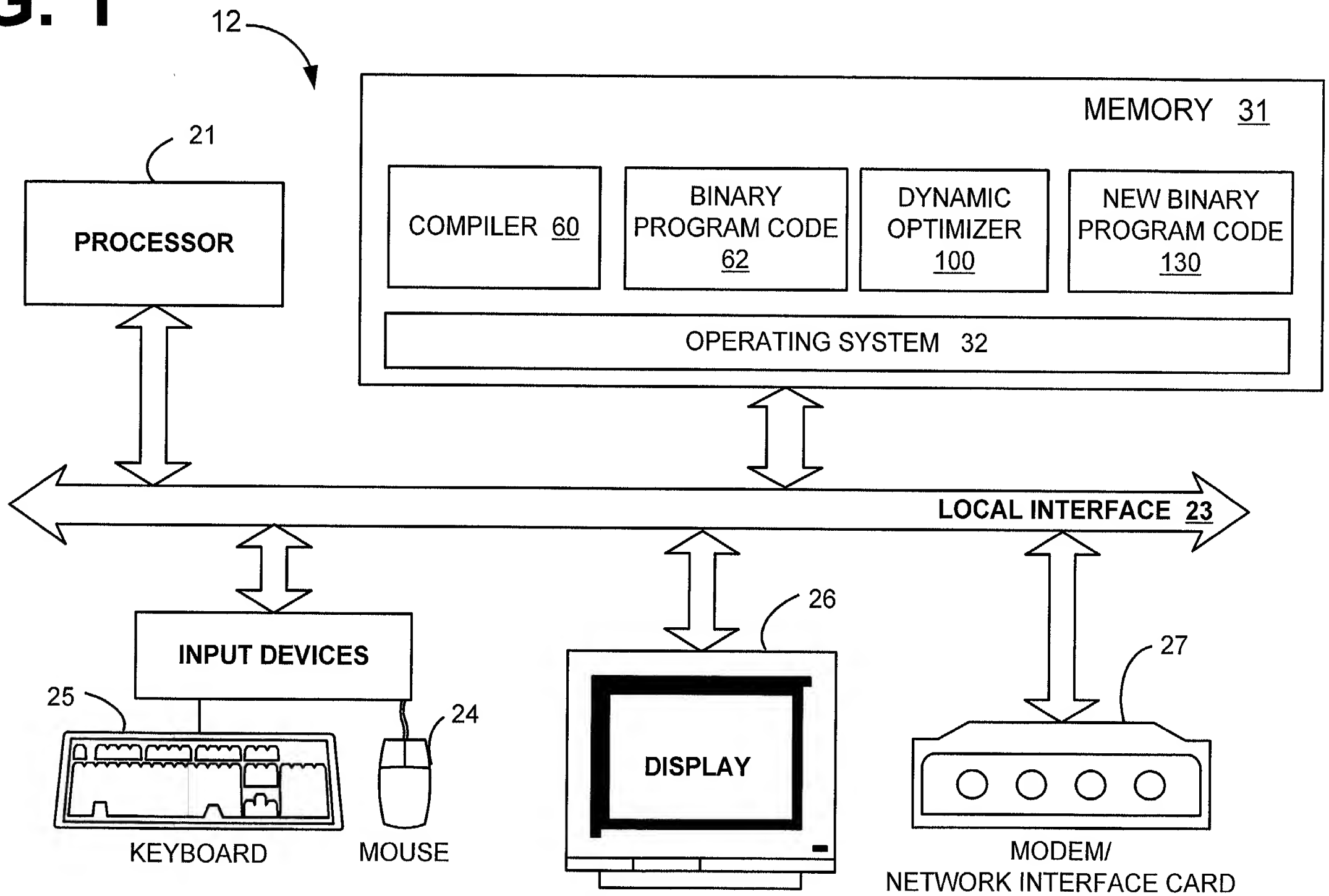

12
21
PROCESSOR
MEMORY 31
COMPILER 60
BINARY PROGRAM CODE 62
DYNAMIC OPTIMIZER 100
NEW BINARY PROGRAM CODE 130
OPERATING SYSTEM 32
LOCAL INTERFACE 23
INPUT DEVICES
25
24
KEYBOARD
MOUSE
26
DISPLAY
27
MODEM/
NETWORK INTERFACE CARD

Fig. 2A

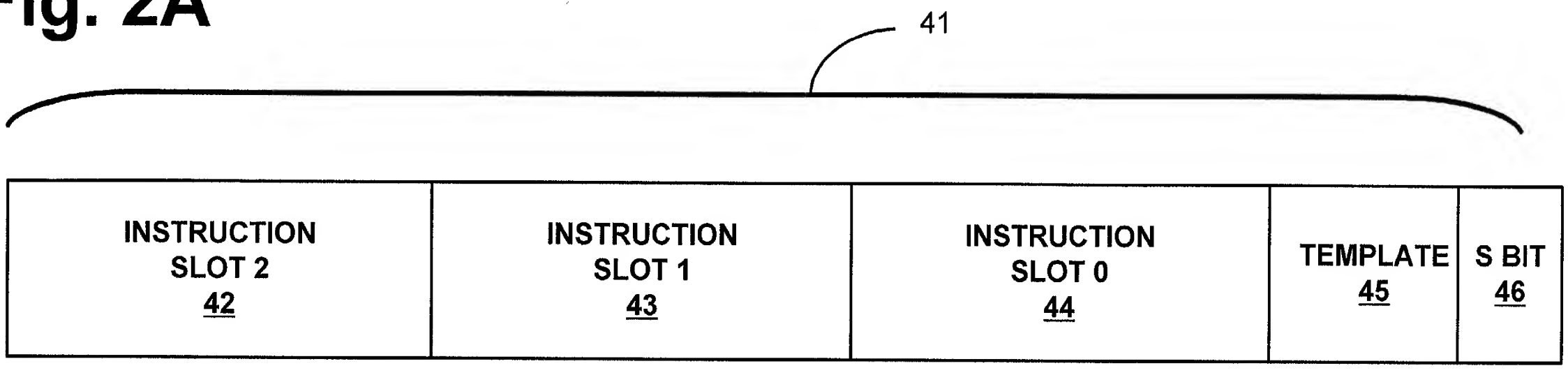

41
INSTRUCTION SLOT 2 42
INSTRUCTION SLOT 1 43
INSTRUCTION SLOT 0 44
TEMPLATE 45
S BIT 46

Fig. 2B

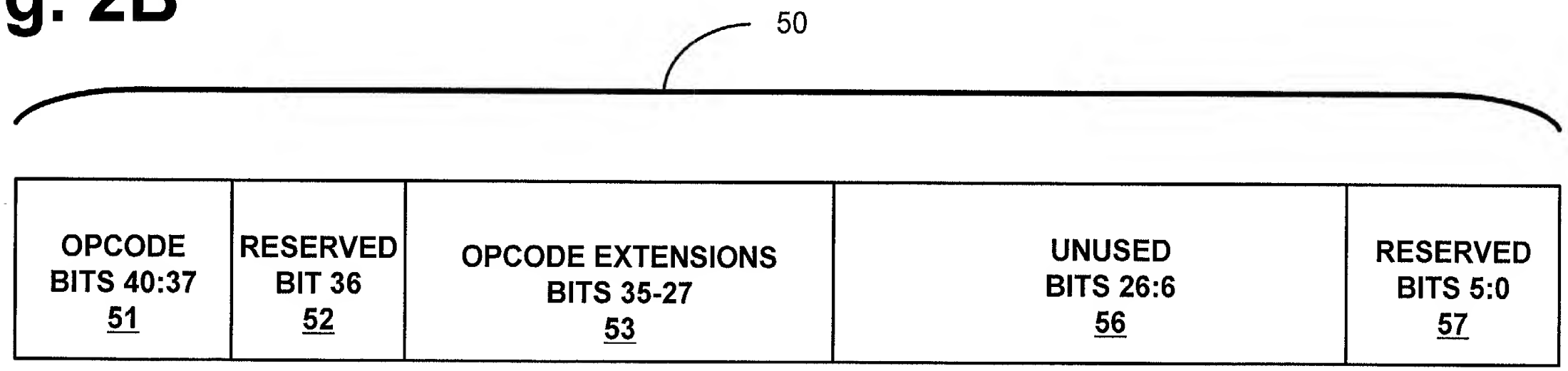

50
OPCODE BITS 40:37 51
RESERVED BIT 36 52
OPCODE EXTENSIONS BITS 35-27 53
UNUSED BITS 26:6 56
RESERVED BITS 5:0 57

FIG. 3A

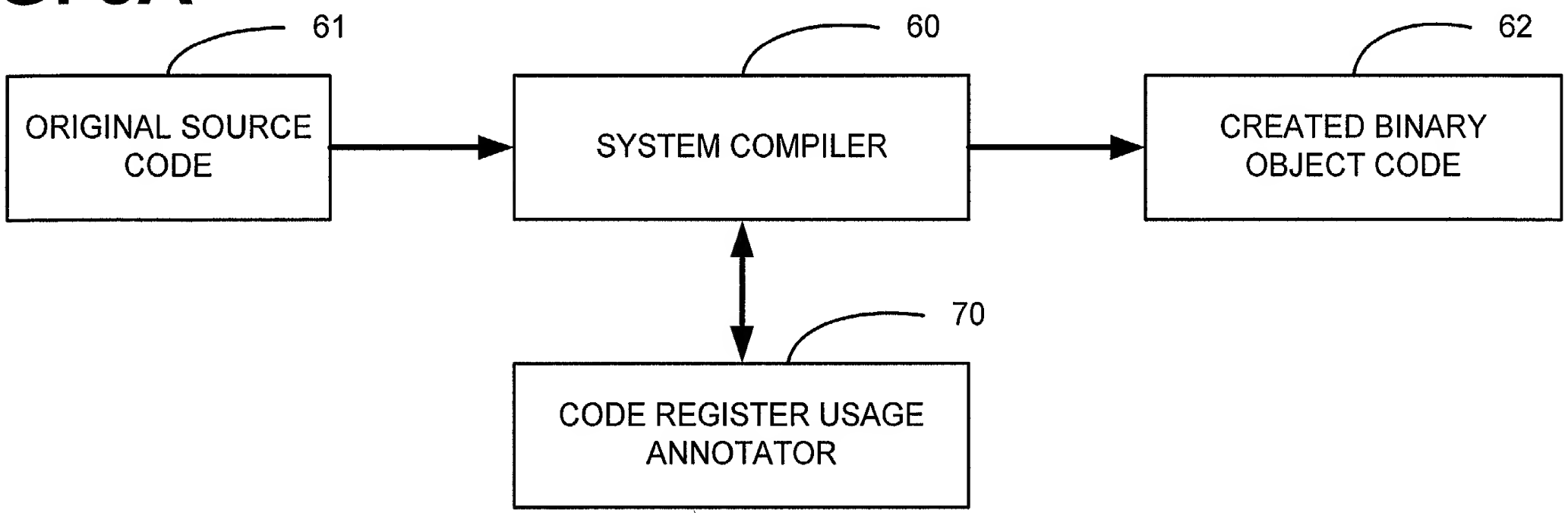

61
60
62
ORIGINAL SOURCE CODE
SYSTEM COMPILER
CREATED BINARY OBJECT CODE
70
CODE REGISTER USAGE ANNOTATOR

FIG. 3B

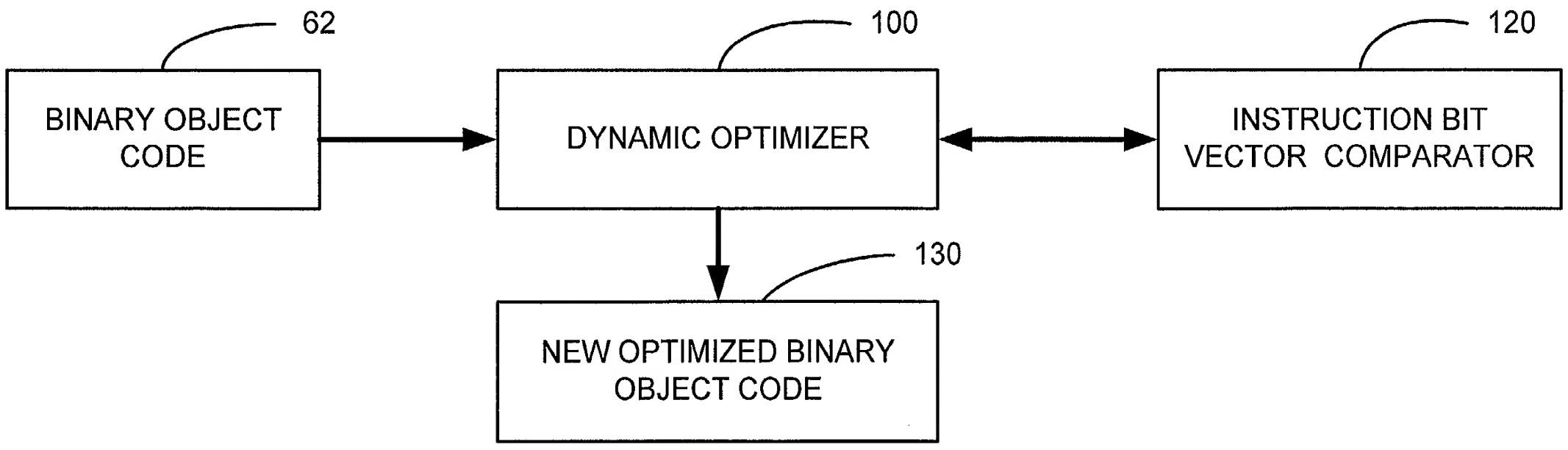

62
100
120
BINARY OBJECT CODE
DYNAMIC OPTIMIZER
INSTRUCTION BIT VECTOR COMPARATOR
130
NEW OPTIMIZED BINARY OBJECT CODE

FIG. 4

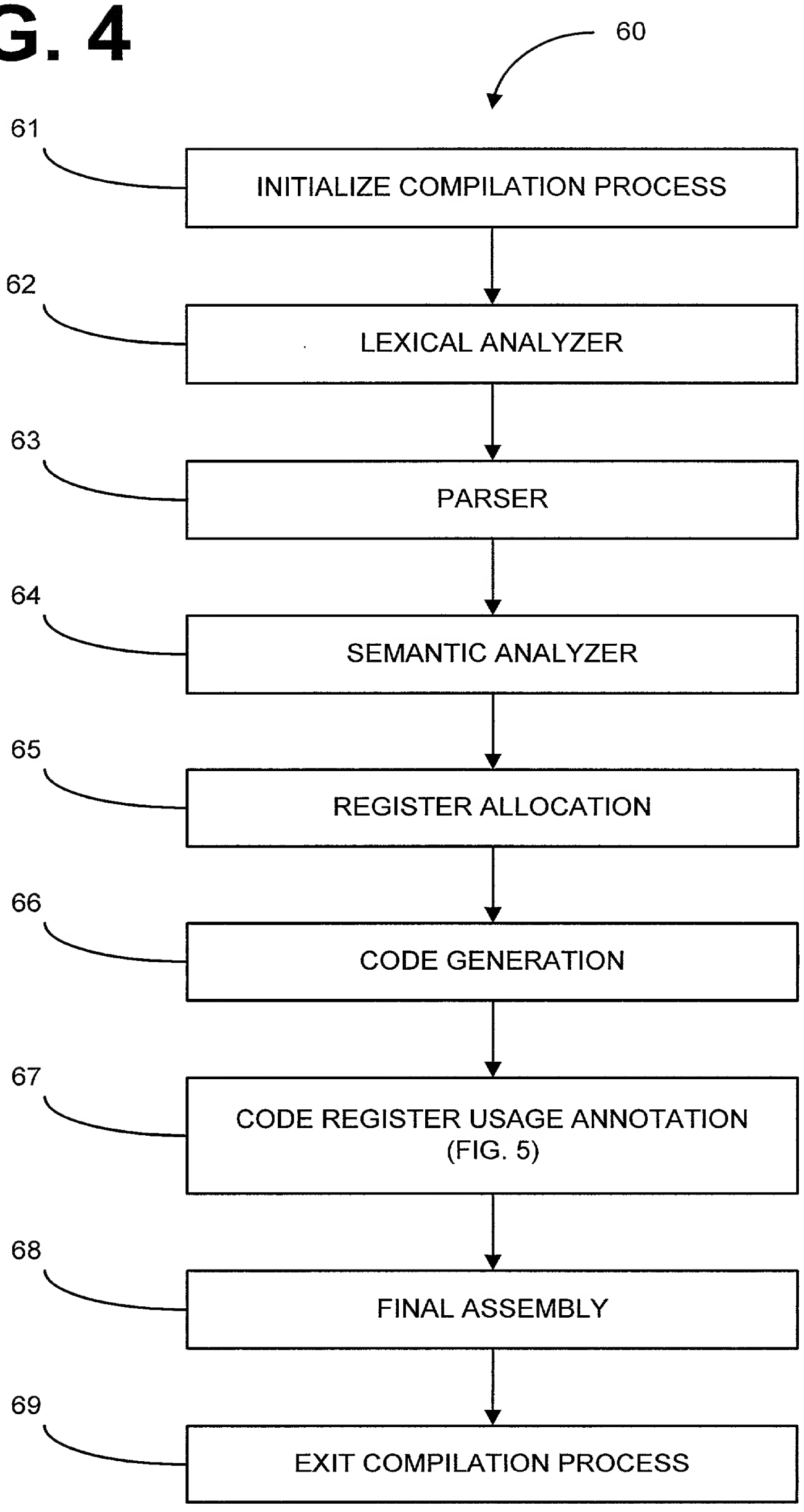

60
61
INITIALIZE COMPILATION PROCESS
62
LEXICAL ANALYZER
63
PARSER
64
SEMANTIC ANALYZER
65
REGISTER ALLOCATION
66
CODE GENERATION
67
CODE REGISTER USAGE ANNOTATION
(FIG. 5)
68
FINAL ASSEMBLY
69
EXIT COMPILATION PROCESS

FIG. 5

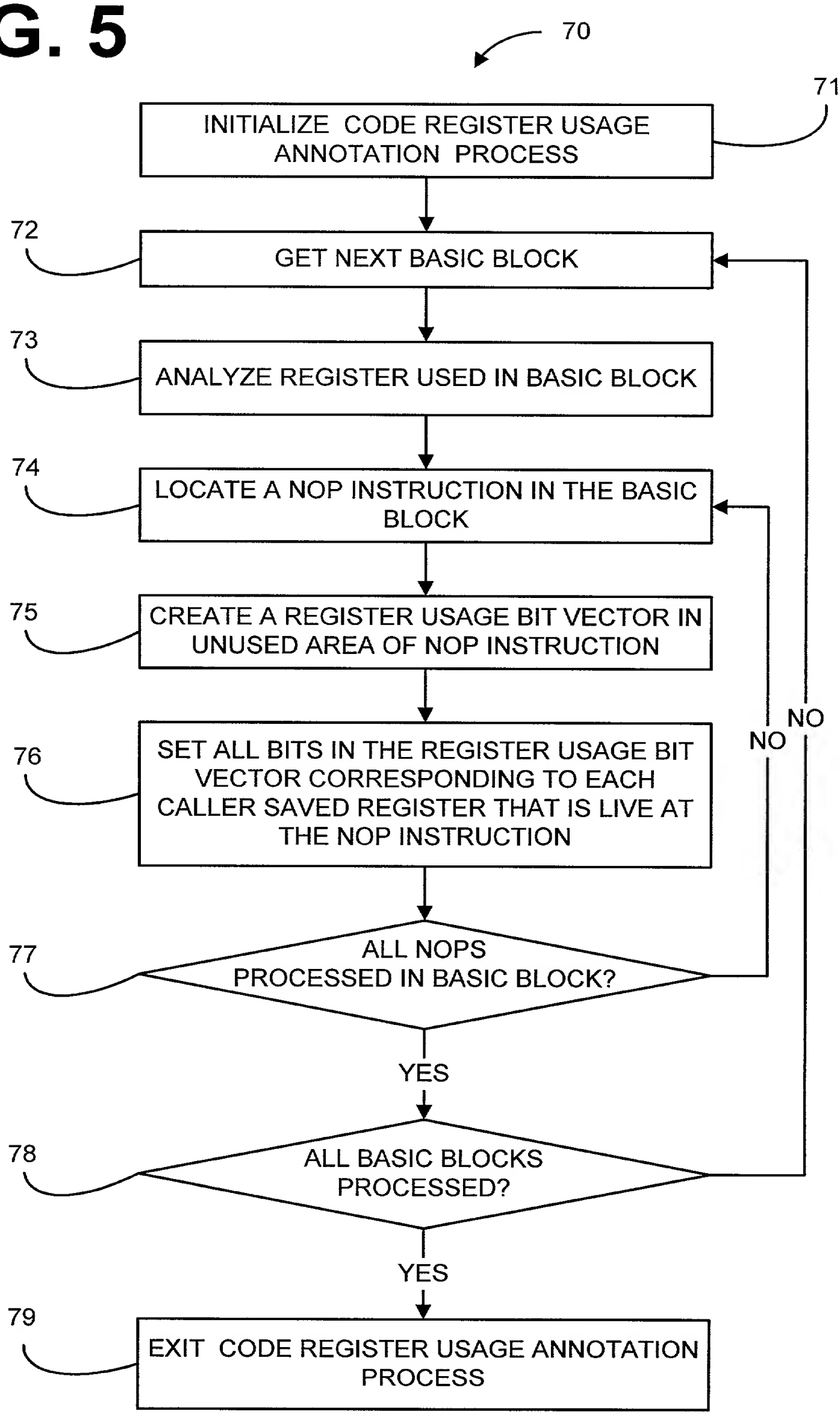

70
71
INITIALIZE CODE REGISTER USAGE ANNOTATION PROCESS
72
GET NEXT BASIC BLOCK
73
ANALYZE REGISTER USED IN BASIC BLOCK
74
LOCATE A NOP INSTRUCTION IN THE BASIC BLOCK
75
CREATE A REGISTER USAGE BIT VECTOR IN UNUSED AREA OF NOP INSTRUCTION
76
SET ALL BITS IN THE REGISTER USAGE BIT VECTOR CORRESPONDING TO EACH CALLER SAVED REGISTER THAT IS LIVE AT THE NOP INSTRUCTION
77
ALL NOPS PROCESSED IN BASIC BLOCK?
NO
YES
78
ALL BASIC BLOCKS PROCESSED?
NO
YES
79
EXIT CODE REGISTER USAGE ANNOTATION PROCESS

Fig. 6

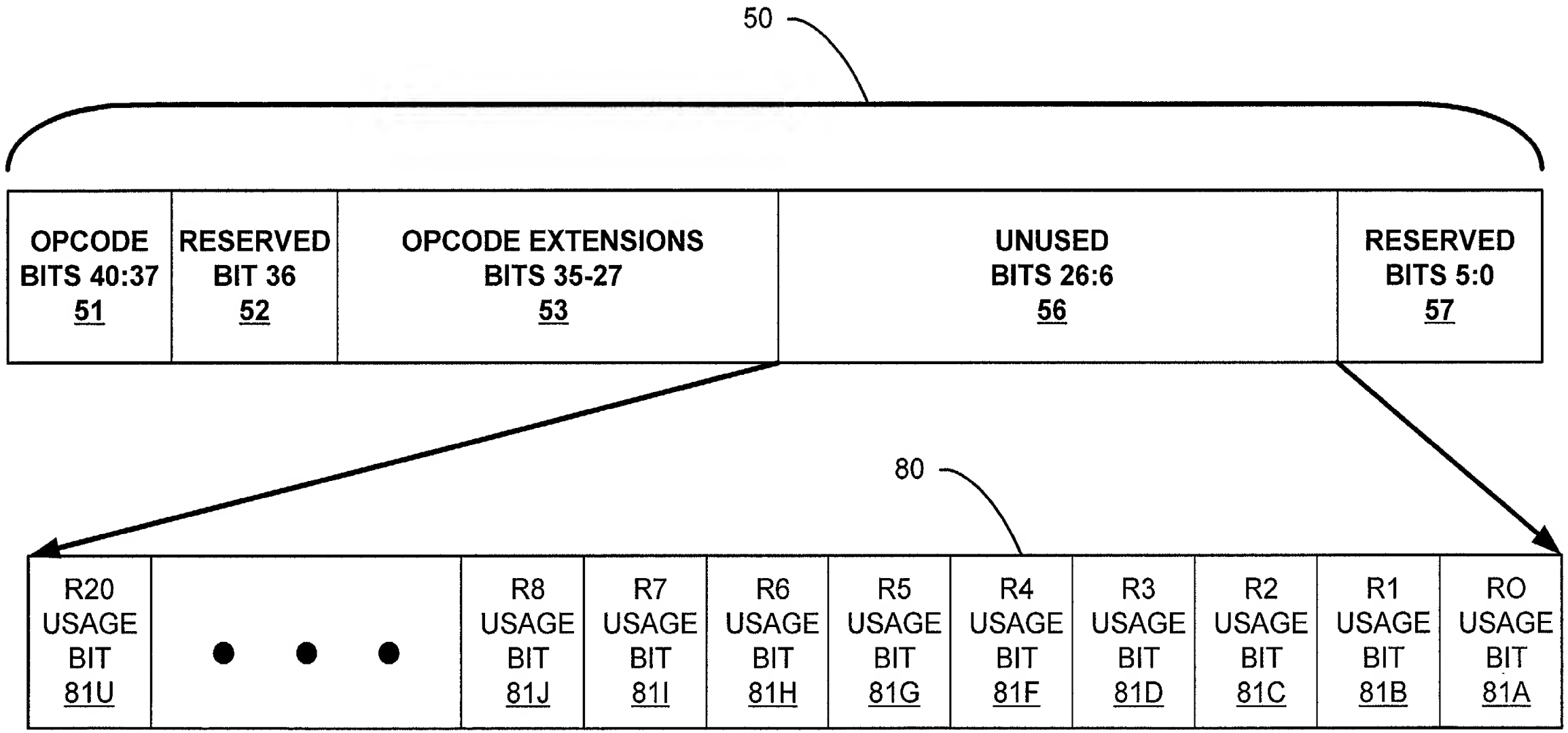

50
OPCODE BITS 40:37 51
RESERVED BIT 36 52
OPCODE EXTENSIONS BITS 35-27 53
UNUSED BITS 26:6 56
RESERVED BITS 5:0 57
80
R20 USAGE BIT 81U
R8 USAGE BIT 81J
R7 USAGE BIT 81I
R6 USAGE BIT 81H
R5 USAGE BIT 81G
R4 USAGE BIT 81F
R3 USAGE BIT 81D
R2 USAGE BIT 81C
R1 USAGE BIT 81B
RO USAGE BIT 81A

FIG. 7

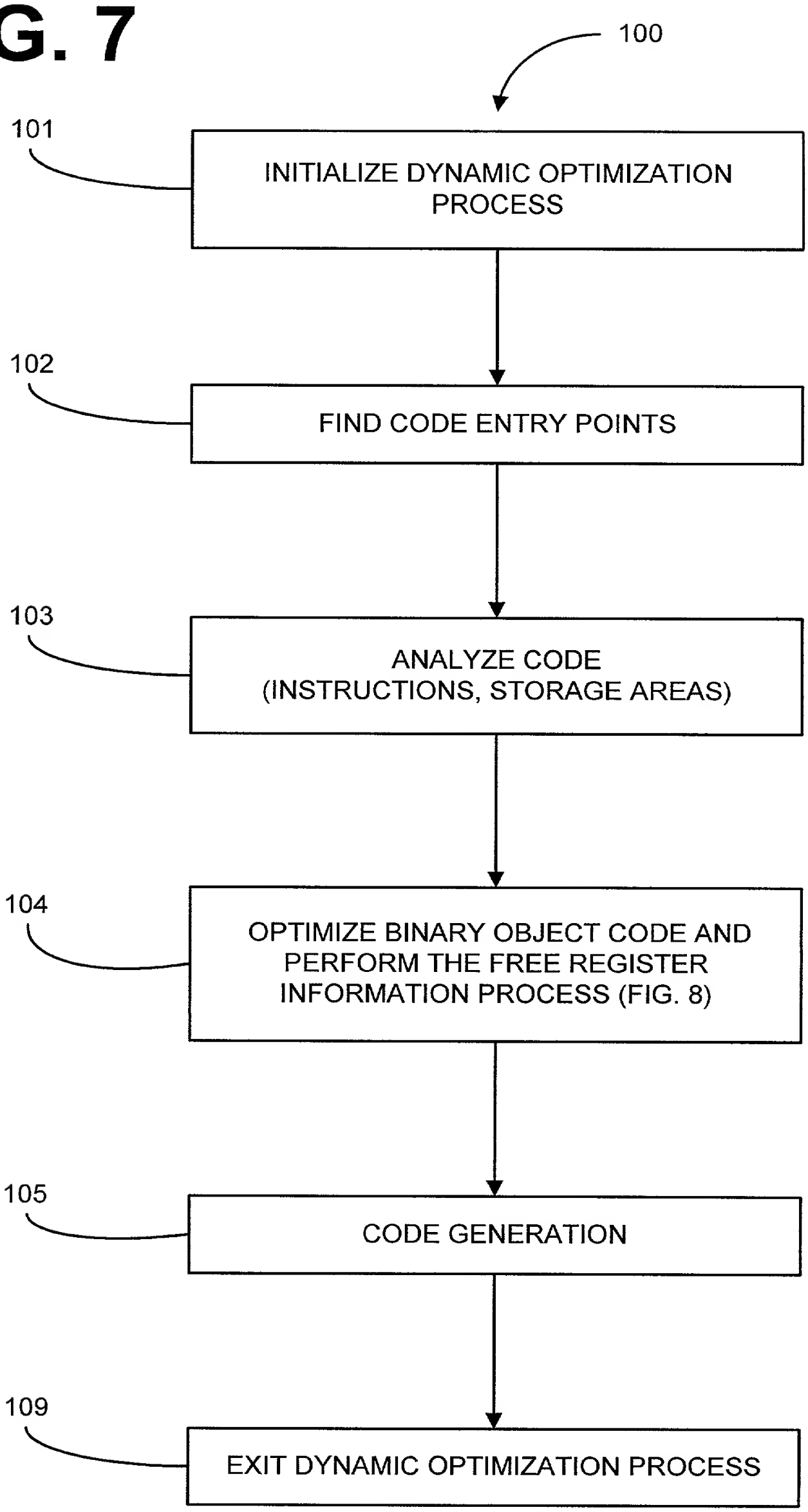

100
101
INITIALIZE DYNAMIC OPTIMIZATION PROCESS
102
FIND CODE ENTRY POINTS
103
ANALYZE CODE (INSTRUCTIONS, STORAGE AREAS)
104
OPTIMIZE BINARY OBJECT CODE AND PERFORM THE FREE REGISTER INFORMATION PROCESS (FIG. 8)
105
CODE GENERATION
109
EXIT DYNAMIC OPTIMIZATION PROCESS

FIG. 8

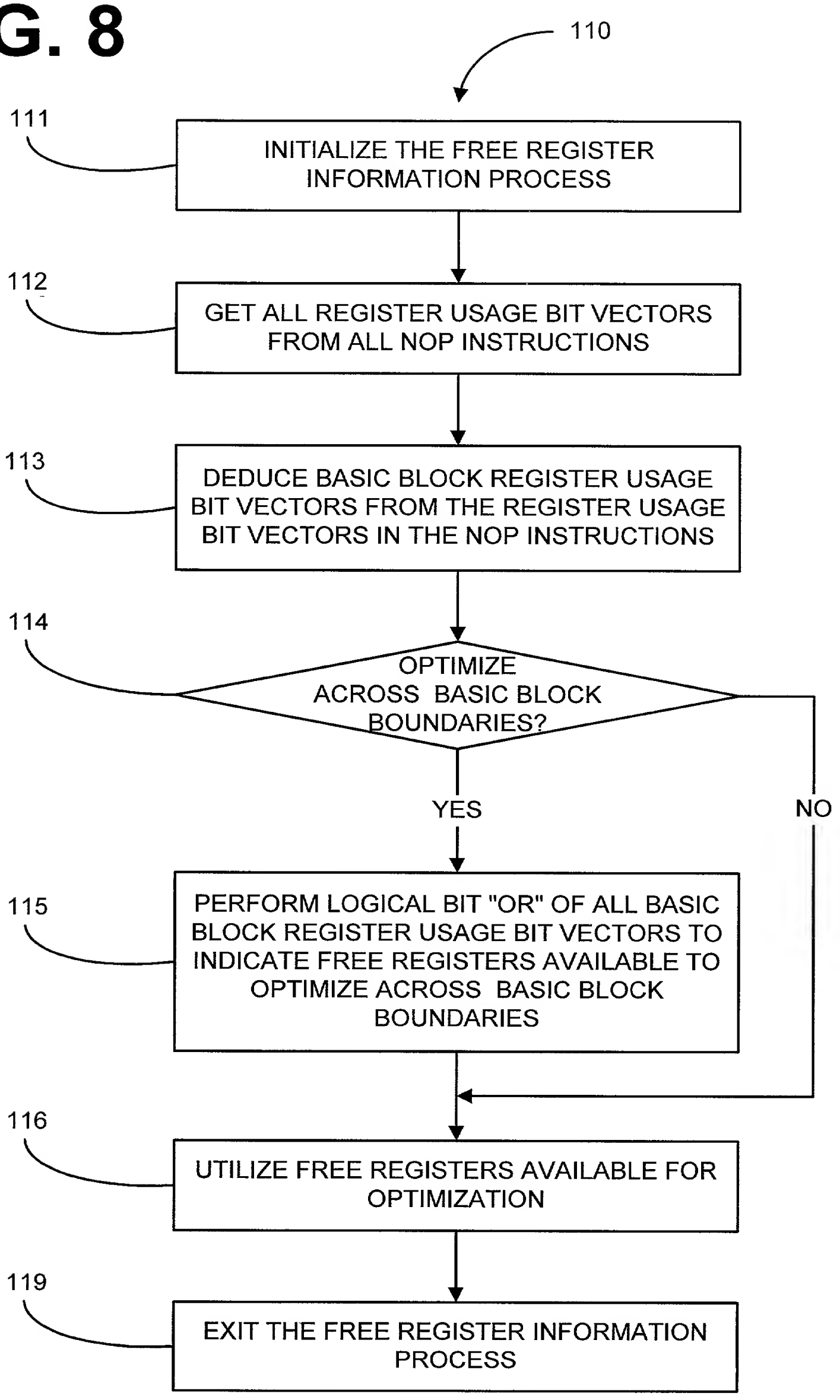
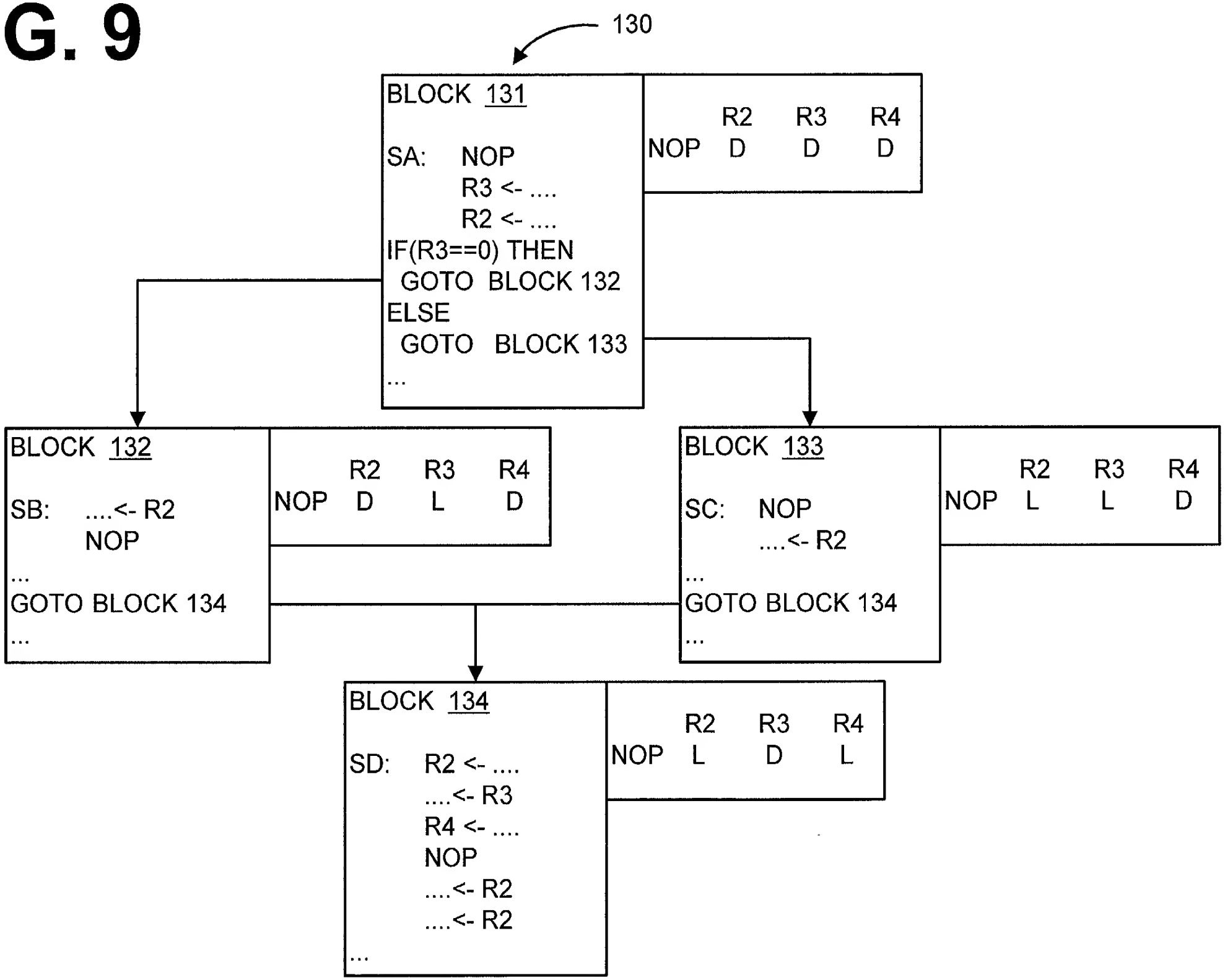

110
111
INITIALIZE THE FREE REGISTER INFORMATION PROCESS
112
GET ALL REGISTER USAGE BIT VECTORS FROM ALL NOP INSTRUCTIONS
113
DEDUCE BASIC BLOCK REGISTER USAGE BIT VECTORS FROM THE REGISTER USAGE BIT VECTORS IN THE NOP INSTRUCTIONS
114
OPTIMIZE ACROSS BASIC BLOCK BOUNDARIES?
YES
NO
115
PERFORM LOGICAL BIT "OR" OF ALL BASIC BLOCK REGISTER USAGE BIT VECTORS TO INDICATE FREE REGISTERS AVAILABLE TO OPTIMIZE ACROSS BASIC BLOCK BOUNDARIES
116
UTILIZE FREE REGISTERS AVAILABLE FOR OPTIMIZATION
119
EXIT THE FREE REGISTER INFORMATION PROCESS

FIG. 9



DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION — ATTORNEY DOCKET NO. 10991766-1

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

SYSTEM AND METHOD FOR EFFICIENTLY PASSING INFORMATION BETWEEN COMPILER AND POST-COMPILE-TIME SOFTWARE

the specification of which is attached hereto unless the following box is checked:

( ) was filed on _____ as US Application Serial No. or PCT International Application Number _____ and was amended on _____ (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35 U.S.C. 119
			YES: ___ NO: ___
			YES. ___ NO: ___

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION SERIAL NUMBER	FILING DATE

U. S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION SERIAL NUMBER	FILING DATE	STATUS (patented/pending/abandoned)

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) listed below to prosecute this application and transact all business in the Patent and Trademark Office connected therewith.

Christine H. Smith	Jeffery B. Fromm	Douglas M. Gilbert	Denise Lee
Reg. No. 43,133	Reg. No. 30,558	Reg. No. 27,196	Reg. No. 35,931

Send Correspondence to:
IP Administration
Legal Department, 20BN
HEWLETT-PACKARD COMPANY
P.O. Box 10301
Palo Alto, California 94303-0890

Direct Telephone Calls To:
Christine H. Smith, Esq.
(408) 447-4764

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Inventor: Ding-Kai Chen **Citizenship:** USA

Residence: 6840 Chiala Ln., San Jose, CA 95129

Post Office Address: 6840 Chiala Ln., San Jose, CA 95129

Ding-Kai Chen — 10/15/99

Inventor's Signature **Date**

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION (continued) ATTORNEY DOCKET NO. 10981788-1

Full Name of # 2 joint inventor: Dz-Ching Ju Citizenship: USA

Residence: 1078 Robbia Dr., Sunnyvale, CA 94087

Post Office Address: 1078 Robbia Dr., Sunnyvale, CA 94087

Dz-Ching Ju Oct 21, 1999

Inventor's Signature Date

Full Name of # 3 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date

Full Name of # 4 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date

Full Name of # 5 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date

Full Name of # 6 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date

Full Name of # 7 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date

Full Name of # 8 joint inventor: Citizenship:

Residence:

Post Office Address:

Inventor's Signature Date